Projet 1 : un analyseur lexical en Python

Thibault PENNING

Un analyseur lexical (appelé aussi lexer) est un programme qui sépare en une liste de symbole (appelé token ou lexème). Ce genre de programme est en général utilisé lors de la première phase de compilation, ou pour réaliser une coloration syntaxique.

L'analyseur

La première étape de ce projet consiste à réaliser l'analyseur en lui-même. Il n'est pas attendu de votre part de réaliser tout l'analyseur, et il est conseiller de procéder étape par étape. Un code propre, sans erreurs, clair et commenté sera toujours préférable son inverse, même si moins complet.

L'analyseur sera composé d'une fonction `analyse` qui prendra en entrée une chaine de caractère contenant du code Python et produira en sortie une liste de tuple, où chaque tuple contient en premier le type du lexème, ainsi que la chaine de caractère qui correspond à ce lexème.

```python
from typing import List, Tuple, Any

def analyse(programme:str)->List[Tuple[Any, str]]:
    pass
```

Exemple :

Le type de donnée représentant le type du token est à votre choix. Je vous conseille de choisir un string (comme dans l'exemple) ou un Enum (hors programme).

Parmi les tokens qui peuvent être détecté voici une liste (non exhaustive mais trop complète, il n'est pas nécessaire de tout réaliser encore une fois):

1. Lire les littéraux simples (entier, flottant et binaire)
2. Lire les opération (une liste est disponible dans la documentation Python, n'utilisez que ceux que vous connaissez)

Listing 1 Exemple de sortie possible d'analyse

```
>>> analyse("a = 1")
[("name", "a"), ("space", " "), ("operator", "="), ("int", "1")]

>>> analyse("""
for c in 'Hello World !':
    print(c)""")

[("keyword", "for"), ("space", " "), ("name", "c"), ("space", " "), ("keyword"  "in"), ("str
```

3. Lire les commentaire
4. Lire les mots clefs (une liste est disponible dans la documentation Python(dans la documentation Python), n'utilisez que ceux que vous connaissez)
    a) Lire les boucle (for et while) et condition (if)
    b) Lire les fonctions
5. Lire les arguments
6. Ajouter les littéraux chaine de caractère
7. Les méthodes (exemple dans `["a", "b"].append("c")`, append est une méthode)

Pour partir sur une bonne base, je vous conseille la chose suivante : créer une fonction pour chaque type de lexème (exemple une qui détecte les mots clef, une les entiers, …) et qui retourne soit "Je ne peux pas en faire en lexème", soit "Je peux en faire un lexème mais il en faut plus", soit "Voici le lexème que j'ai pu faire". Une autre possibilité est que chaque fonction retourne le plus grand lexème qu'elle est capable de produire.

Une boucle parcourra simplement le programme, caractère par caractère, et ajoutera chaque caractère à un possible lexème. Si celui-ci trouve un lexème et l'accepte, alors il ajoute ce lexème dans la liste, réinitialisé le possible lexème avec la lettre suivant et continue.

L'architecture sous forme de fonction est particulièrement utile car modulaire, contrairement à une architecture monolithique. Vous serrez possiblement amené à changer les fonctions (et notamment ce qu'elle retourne) au besoin.

Attention aux priorités. Ainsi `"def"` est bien un `string` et non pas un mot-clef entouré de `"`. Ainsi on ne doit pas accepter `def` comme un mot-clef tant que `string` peut encore créer un lexème. Ou encore `assert` doit être détecté comme mot-clef et non comme le mot-clef `as` suivie de `sert`.

Il peut être utile de regarder la taille de lexème renvoyer. Ainsi `print(a)` pourra détecter `print` comme un `name` car il possède est suivie d'une `(`. La boucle reprendre donc à partir de `(` et non à `a` si on ne fait pas attention.

> **Important**
>
> Il est nécessaire de garder tous les caractères du programme, afin qu'il puisse être reconstruit plus tard

N'oubliez pas, la pilosophie KISS est souvent ce qu'il y a de mieux pour produire du "bon code".

Coloration syntaxique

Le but est de créer une fonction `coloration` qui prend en entrée un chemin vers un fichier Python, et génère un fichier HTML représentant ce code Python avec une coloration syntaxique.

```python
def coloration(file_name:str)->None:
    with open(file_name, "r") as input_file:
        pass

    with open(output_file_name, "w") as output_file:
        pass
```

Pour cela vous utiliserez la fonction `analyse`, copierez le lexème dans le fichier HTML et changerez la couleur en fonction du type du lexème. N'ayez crainte, cette fonction est plus simple qu'il n'y paraît, toute l'intelligence se situe dans la fonction analyse.

> **Indice**
>
> Essayer cette balise HTML :
>
> ```html
> <font color="blue">for</font>
> ```

Pour un meilleur rendu, vous pouvez utiliser une police dite "monospace".

Dérouler

Par **groupe de 2 ou 3**.

S'il vous plait faites des **groupes mixtes en niveau**. Toutes les personnes doivent participer au projet. **L'investissement est un part de la note**. Ainsi il est conseillé au plus expérimenté de ne pas être toujours derrière le clavier, ou de dicter, mais d'être une aide technique pour

les autres. De la même manière que ne pas participer est dévalorisé, ne pas laisser les autres s'exprimer seras aussi dévalué.

Les rendus attendus sont les suivants :

1. Un fichier **README.md** (ou autre), décrivant le programme brièvement, son fonctionnement brièvement et surtout les entrée et sortie : comment exécuter votre programme (arguments, fonction utilisable en API, …) qui me permettent d'exécuter votre code.
2. Un fichier **SPECS.md** décrivant vos fonctions utilisés et SURTOUT les lexèmes utilisés, le type dans les tuples etc, comme dans l'exemple Listing 1. Cela pourra se comporter comme un tableau, avec le type de lexème, l'organisation du tuple, et un exemple.
3. Un fichier, sous quelconque forme que vous souhaitez, me permettant d'apprécier le **travail individuel de chacun et l'organisation du groupe**. Cela peut inclure un cahier de projet, un Gantt, un tableau des taches format Kaban, …
4. Les **fichiers en Python* (vous pouvez viser Python 3.11) sources qui permettent le fonctionnement de votre projet.

En plus de la production de code, il sera attendu des élèves une explication sous forme d'une **présentation orale**, du fonctionnement de leur projet et de la raison derrière leur choix. Les erreurs qu'ils ont faites face est une part non négligeable dans un projet, ainsi présenter les erreurs (résolu ou non) avec une explication est possiblement tout aussi valorisant qu'une explication d'un choix qui fonctionne.

Tout code de teste susceptible d'être laissé, s'il permet de s'assurer du bon fonctionnement du programme (cela est même valorisé).

Tout autres documents servant la compréhension du projet (cahier des charges, schéma, …) sera valorisé.

> **!** Important
>
> De nombreux analyseur lexical existe, et de nombreuse libraire pour Python ont donc été développés. Il est évident que vous ne devez pas les utiliser. De même, lors de vos recherches vous serez amené à croiser des ressources utilisant des expressions régulière ou automate finis. Ce genre d'outils est totalement hors programme (vous serez surement amené à les croisés dans vos études si vous poursuivez en informatique) et ne doit donc pas non plus être utilisé.
> Le but du projet est d'utiliser une boucle for et des fonctions spécialisés afin de réaliser l'analyse.